Functional Encryption: A New Vision for Public Key Cryptography

Dan Boneh Amit Sahai Brent Waters

Encryption is a method for a user to securely share data over an insecure network or storage server. Before the advent of public-key cryptography, a widely held view was that for two users to communicate data confidentially they would need to first establish a mutually held secret key k. While this might be acceptable for some small or tightly knit organizations, such a solution was clearly infeasible for larger networks such as today's Internet. Over thirty years ago, Diffie and Hellman [DH76a, DH76b] put forth the concept of public-key cryptography, where two parties can securely communicate with each other *without* having a prior mutual secret, radically challenging the conventional wisdom of the time.

Today public key encryption is an invaluable tool and its use is ubiquitous in securing web communication (e.g. HTTPS and SSH), voice traffic, and storage systems. However, there is an ingrained view that:

- Access to the encrypted data is "all or nothing" – one either decrypts the entire plaintext or learns nothing at all about the plaintext (other than a bound on its length), and
- Encryption is a method to encode data so that a *single* secret key can decrypt that data.

For many applications, however, this notion of public-key encryption is insufficient. For example, the encryptor may want to encrypt data so that anyone who satisfies a certain policy can then decrypt. Consider encrypting a message to a company so that the only users who can decrypt are:

> employees in the accounting or sales departments whose office is in the main building.

Realizing this using existing public-key encryption raises several difficulties:

- How do we discover the public keys of all individuals who satisfy this policy?
- What if someone joins the system or receives certain credentials well after the data is encrypted and stored?
- What if we want to give someone a partial view of the plaintext depending on their credentials?
- Should one even be allowed to learn the identities of all individuals who have certain credentials?

Functional encryption. We believe that it is time to adopt a new broad vision of encryption that takes into account situations and concerns like those outlined above. To this end, we advocate the concept of *functional encryption* where a decryption key enables a user to learn a specific *function* of the encrypted data and nothing else. Briefly, in a functional encryption system there is a *trusted authority* who holds a master secret key known only to this authority. When the authority is given the description of some function f as input, it uses its master secret key to generate a derived secret key $\mathrm{sk}[f]$ associated with f. Now anyone holding $\mathrm{sk}[f]$ can compute $f(x)$ from an encryption of any x. In symbols, if $E(\mathrm{pk}, x)$ is an encryption of x then decryption accomplishes the following:

$$\text{given } E(\mathrm{pk}, x) \text{ and } \mathrm{sk}[f], \quad \text{decryption outputs } f(x).$$

Note that it is $f(x)$ that is made available to the secret key holder, even though x was the value that was encrypted. A functional encryption system can support a variety of functions in this fashion. Intuitively, the security of the functional encryption system should guarantee that the secret key holder can learn nothing more about x beyond $f(x)$. In this way, we envision functional encryption as being analogous to secure computation [Yao82, GMW87], but with the critical difference that functional encryption is completely non-interactive once a recipient has obtained her secret key.

Let us consider what could be achieved if we could realize functional encryption for a broad set of functions:

Spam Filtering on Encrypted Mails: A user wishes to leverage a partially trusted proxy to filter out all encrypted emails that are identified as spam according to the user's criteria. We would like to achieve the seemingly conflicting goals of hiding the email's contents from the proxy while allowing the proxy to determine if the email is spam according to some arbitrary criteria. The user can achieve this by setting up a functional encryption system and then giving the proxy a key $\mathrm{sk}[f]$ where f is the user specified program that outputs 1 if the plaintext is spam and 0 otherwise. The proxy can use $\mathrm{sk}[f]$ to test if an encrypted email is spam without learning anything more about the plaintext, as shown in Figure 1.

Naturally, one can consider generalizations of this idea. For instance the proxy might selectively send important email (as deemed by the function f) to the user's mobile device. Taking things further we could imagine that the destination of a packet is encrypted and the secret key $\mathrm{sk}[f]$ allows a router to learn the next hop and nothing more.

Expressive Access Control: In large organizations one will often think of sharing data according to some access policy. In addition to our corporate example, this might occur in other domains such as health care, insurance companies, government institutions and universities. Bridging the gap between how one thinks of sharing

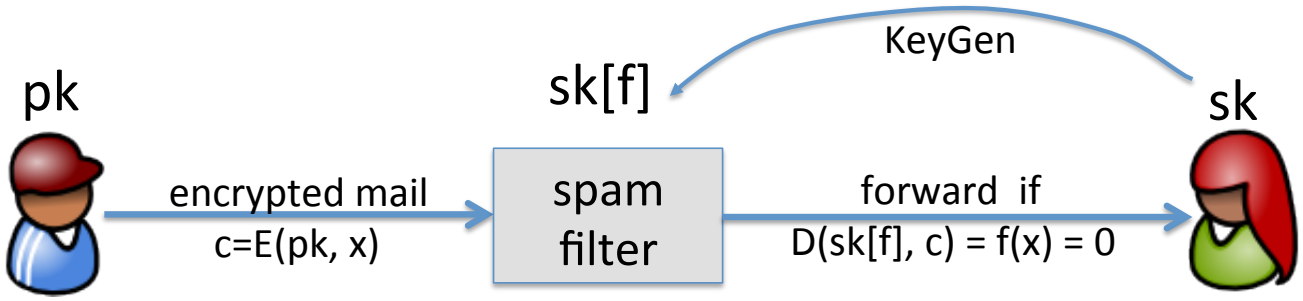

Figure 1: The email recipient, who has a master secret key sk, gives a spam filtering service a key sk$[f]$ for the functionality f. This f satisfies $f(x) = 1$ whenever message x is marked as spam by a specific spam predicate; otherwise $f(x) = 0$. A sender encrypts an email x to the recipient, but the spam filter blocks the email if it is spam. The spam filter learns nothing else about the email contents.

data and discovering the public keys of all users who match or will match this can be difficult and is subject to all the problems stated above. For example, one might try to encrypt data separately to the public key of every user that matches a certain policy. However, as noted above this requires identification of each user as well as the overhead of encrypting to each one individually. Moreover, this does not cover users that don't meet the criteria today, but will in the future.

Using functional encryption one can directly express how one wishes to share the data in the encryption process itself. In particular, one can encrypt $x = (P, m)$ where m is the data one wishes to share and P is the access policy that describes how they want to share it. Then a user's secret key function sk$[f]$ will check whether a user's credentials or attributes match this policy and only reveal m in this case. Corresponding to the example mentioned earlier, P could embed the policy ("ACCOUNTING" OR "SALES") AND "MAIN BUILDING". A recipient's function f would embed the attributes of the particular user and check if these satisfy the formula and if so return m.

Mining Large datasets: Data mining is used in medical research, social networks, network security, and financial fraud detection to name a few. Often we want to give users the ability to mine datasets for certain types of queries, but not let them learn anything beyond that. Consider a medical researcher who wants to test if there is a link between a genotype and a type of cancer in an ethnic group. If we have data consisting of patient gene sequences and medical history, we would like to give the researcher the ability to test for this linkage, without revealing the details of all patients' medical conditions to the researcher.

It is also important to note that in practice, we typically will not know the queries that will be of interest until well after the data is created and stored. Functional encryption provides an elegant solution for these types of problems. When data is created it is simply encrypted in a functional encryption system. Later on, a user requests to be be able to learn a certain query or function f of the data. If this is authorized, the user is given sk$[f]$ and can apply this to any existing or future encrypted data. Thus, in a functional encryption system supporting a class of functions $\mathcal{F}$, a user could be given the capability to compute any function from this class on the dataset.

These examples motivate the research agenda that we put forward here: to create functional encryption systems supporting the richest possible families of functions, and understand what limitations are inherent for functional encryption systems.

1. FUNCTIONAL ENCRYPTION

Recall that public key encryption systems, such as RSA or ElGamal, comprise of three algorithms:
(1) a setup algorithm that outputs a secret key denoted sk and a public key pk – anyone can encrypt message using pk, but only the secret key holder can decrypt,
(2) an encryption algorithm E that takes a public key pk and a message as input and outputs a ciphertext, and
(3) a decryption algorithm D that takes a secret key sk and a ciphertext as input and outputs a message.

A functional encryption system comprises the same three algorithms, but also contains a fourth algorithm called KeyGen. Here the secret key output by the Setup algorithm is called the master key and is denoted by mk. The KeyGen algorithm takes as input mk and the description of some function f. It outputs a key that is specific to the function f and denoted sk$[f]$. More precisely, if c is the result of encrypting data x with public key pk then

$$D(\text{sk}[f], c) \quad \text{outputs} \quad f(x)$$

We emphasize that sk$[f]$ does not fully decrypt c. It only outputs a function f of the full decryption. To fully decrypt c one can use a secret key sk$[g]$ where g is the identity function, namely $g(x) = x$ for all x.

Informally, security of a functional encryption system means that an attacker who has a set of secret keys sk$[f_1], \ldots,$ sk$[f_\ell]$ can learn nothing about the decryption of some ciphertext c other than what is revealed by the keys at his disposal. We discuss security in more detail in the next section.

To illustrate the power of functional encryption we show how it naturally captures many advanced encryption concepts in cryptography. First, it should be clear that traditional public-key encryption is a very special case of functional encryption where the only supported function is the identity function. The decryptor either learns the complete decryption or nothing at all.

Identity Based Encryption. A more advanced public-key concept called *Identity Based Encryption* or IBE for short, is an encryption system where any string can serve as a public key: a user's email address, a date, an IP address, a location, and even the numbers $1, 2$, and 3 are all public keys. IBE public keys are often called identities and denoted by id. To obtain the secret key for a particular identity the user communicates with an authority who holds a master key. The authority verifies that the user is authorized to receive the requested secret key and if so it generates the secret key using its master key. IBE was first proposed by Adi Shamir [Sha84] in 1984 and the first implementations of IBE were proposed by

Boneh and Franklin [BF01] and Cocks [Coc01] in 2001. Other notable constructions include [BB04, Wat05, Gen06, GPV08, Wat09, ABB10].

Using the terminology of functional encryption we can recast the IBE problem as an equality testing functionality. Let pk and mk be the output of the functional encryption setup algorithm. To encrypt a message m to identity id the encryptor calls the encryption algorithm as

$$E(\mathrm{pk},\ (\mathsf{id}, m)\ )$$

and obtains a ciphertext c. Note that the data being encrypted is the pair (id, m).[1] A recipient with identity id^* obtains a secret key for id^* by asking the authority for a secret key $\mathrm{sk}[f_{\mathsf{id}^*}]$ where the function f_{id^*} is defined as follows[2]

$$f_{\mathsf{id}^*}(\ (\mathsf{id}, m)\ ) := \begin{cases} m & \text{if } \mathsf{id} = \mathsf{id}^*, \\ \perp & \text{otherwise} \end{cases} .$$

The authority generates $\mathrm{sk}[f_{\mathsf{id}^*}]$ using its functional master key mk. Using this secret key the user can decrypt messages intended for identity id^*, but learns nothing about messages encrypted for other identities.

Recall that IBE systems reduce the reliance on certificate directories needed for traditional public-key encryption: to encrypt to identity id the encryptor need only have the global public key pk and the recipient's identity id. General functional encryption systems have the same property — they require no online certificate directory. An encryptor need only have the global public key pk and the payload x being encrypted. It needs no other information about the intended recipient(s).

Attribute-based encryption. Another encryption concept called *Attribute-based encryption* or ABE for short, lets the encryptor specify more abstractly who can decrypt a specific ciphertext. ABE was proposed by Sahai and Waters [SW05] and later refined by Goyal, Pandey, Sahai and Waters [GPSW06] into two different formulations of ABE: Key Policy ABE and Ciphertext-Policy ABE.

In a Ciphertext-Policy ABE system the encryptor specifies a policy φ on recipient attributes that determines who can decrypt the ciphertext. For example, the encryptor can encrypt messages to anyone who is a

("US citizen" and "female") or ("over 30")

which is a boolean formula φ on three variables. To encrypt a message m with decryption policy φ the encryptor calls

$$E(\mathrm{pk},\ (\varphi, m)\ )$$

and obtains a ciphertext c.

Now, consider a recipient who wants to decrypt the ciphertext. The recipient has a number of attributes, say:

"US citizen", "Rhodes Scholar","female", "under 30".

Let n be the total number of attributes and we represent the set of user attributes as a boolean vector of length n – the vector is 1 at positions that correspond to attributes that are true and is 0 everywhere else. With this setup every user has an attribute vector $\mathbf{u}$ in $\{0,1\}^n$.

[1]Using our earlier notation we would have $x = (\mathsf{id}, m)$. However, we will omit x for readability.

[2]We use the $\perp$ symbol as a special symbol to denote failure to decrypt.

A recipient with attribute vector $\mathbf{u}$ obtains a secret key for his attribute vector by asking the authority for a secret key $\mathrm{sk}[f_{\mathbf{u}}]$ where the function $f_{\mathbf{u}}$ is defined as follows:

$$f_{\mathbf{u}}(\ (\varphi, m)\ ) := \begin{cases} m & \text{if } \varphi(\mathbf{u}) = 1, \\ \perp & \text{otherwise} \end{cases} \quad (1)$$

The authority generates $\mathrm{sk}[f_{\mathbf{u}}]$ using its functional master key mk. Using this secret key the user can decrypt ciphertexts where his attributes satisfy the decryption policy, but learns nothing about the decryption of other ciphertexts.

A related concept called *Key-Policy Attribute Based Encryption* places the access policy φ in the key and the vector $\mathbf{u} \in \{0,1\}^n$ in the ciphertext. The secret key $\mathrm{sk}[f_\varphi]$ will decrypt all encryptions $E(\mathrm{pk},\ (\mathbf{u}, m)\ )$ for which $\varphi(\mathbf{u}) = 1$.

2. SECURITY

The previous section described the syntax for a functional encryption scheme where we explained that the scheme is defined by four algorithms: (Setup, KeyGen, E, D). We now turn to constructing functional encryption systems, but before we do so we must first explain what it means for a functional system to be secure. The full definition is a bit technical and here we only give the high level intuition. We refer the reader to [BSW11] for the details.

Roughly speaking, a functional encryption system is secure if an attacker who has a set of secret keys $\mathrm{sk}[f_1], \ldots, \mathrm{sk}[f_\ell]$ can learn nothing about the decryption of some ciphertext c other than what is revealed by the keys at his disposal. If c is the encryption of some data x, then the attacker can use his secret keys to learn $f_1(x), \ldots, f_\ell(x)$. However, he should be unable to learn anything else about x. For example, if the attacker has secret keys that reveal the first three bits of x, then clearly the attacker can learn these bits given an encryption of x, but he should be unable to learn anything about remaining bits of x.

To give a bit more detail about the security requirements, let $\mathcal{A}$ be a polynomial time adversary that takes as input three things: the public key pk, a set of secret keys $\mathrm{sk}[f_1], \ldots, \mathrm{sk}[f_\ell]$ for functions $f_1, \ldots, f_\ell$ of its choice, and a ciphertext $c = E(\mathrm{pk},\ x)$. This $\mathcal{A}$ might output some information about the decryption of c, such as the least significant bit of x. We say that the system is secure if for every such $\mathcal{A}$ there is a another polynomial time algorithm $\mathcal{B}$, called a *simulator*, that given pk and $f_1(x), \ldots, f_\ell(x)$ — but not given c — is able to output the same information about x that $\mathcal{A}$ output. Since $\mathcal{B}$ never got to see c it must have deduced the information about x strictly from $f_1(x), \ldots, f_\ell(x)$. Since $\mathcal{A}$ and $\mathcal{B}$ output the same information about x, the existence of $\mathcal{B}$ means that the only information $\mathcal{A}$ can learn about x from the ciphertext c is information it can learn from $f_1(x), \ldots, f_\ell(x)$, but cannot learn anything else about x. Hence, $\mathcal{A}$ can learn from c whatever is revealed by the secret keys at its disposal, but cannot learn anything else about m. [3]

Challenge:Preventing Collusion Attacks.

Attacks on functional encryption that make use of multiple functional secret keys are called *collusion* attacks. Preventing these attacks is the main obstacle to constructing secure functional systems. To illustrate the problem consider again the functionality described in (1) and suppose the encryptor wishes to encrypt a message m to the following policy:

"US citizen" and "over 30"

[3]Note that our security model does not rely on any assumptions about trusted hardware or online servers needed during decryption.

A simple implementation is to associate a public key pk_1 with the attribute "US citizen" and a public key pk_2 with the attribute "over 30" and double encrypt the message m as

$$c = E(\text{pk}_1,\ E(\text{pk}_2,\ m\ ))$$

where here $E(\cdot,\cdot)$ is a regular public key encryption algorithm. To decrypt c the recipient — let's call her Alice — must possess both secret keys corresponding to pk_1 and pk_2, thus implementing the conjunction policy specified by the encryptor.

Now, suppose another user, Bob, has attributes

"US citizen" and "male"

where the attribute "male" is associated with a public key pk_3. Bob would be given the secret keys corresponding to pk_1 and pk_3. This would let him decrypt messages encrypted for some policies such as ("US citizen" and "male"). In addition, suppose Alice has the attribute "over 30". Then she is only given the secret key corresponding to pk_2. Thus, she cannot decrypt the message associated with the original policy above on her own.

The problem is that Alice and Bob can collude to combine their secret keys and create new secret keys which neither one should have. For example, Alice and Bob working together can decrypt ciphertexts intended for policy

"over 30" and "male"

even though neither could decrypt this ciphertext by themselves. In this example collusion enabled the users to decrypt a ciphertext that neither should have access to.

Secure constructions.

Secure constructions for complex functionalities must prevent collusion attacks. This is done by binding together all secret keys for a set of attributes so that mixing the keys given to distinct users does not help. As a metaphor, one can imagine that all the keys given to Alice are colored blue while all the keys given to Bob are colored red. Decryption succeeds only when the decryptor uses a set of keys of the same color. The colors ensure that Alice and Bob cannot combine their keys to decrypt ciphertexts they should not.

In practical terms, the colors are implemented using randomization values. All the keys given to Alice are blinded by the same random value while all the keys given to Bob are blinded by a different random value. Decryption with keys that are blinded by the same randomizer produces the correct decrypted message. Decryption with keys blinded by different randomizers results in a random value unrelated to the correct decryption.

3. STATE OF THE ART

The current state of the art in functional encryption can be broken down by considering what information about the plaintext x is exposed by the ciphertext to all participants. We refer to this information as the result of the "empty functionality" denoted $f_\epsilon(\cdot)$. For example, it is inherent in any encryption scheme that the empty functionality expose *some* information about x, such as a bound on the size of the plaintext. When the exact plaintext length is leaked by the ciphertext we write $f_\epsilon(x) = |x|$ to indicate that anyone can learn the plaintext length from the ciphertext.

3.1 Public Index: ABE

In general, we can consider the problem of functional encryption where the data to be encrypted is decomposed into two parts $x = (\mathsf{ind}, m)$, where ind denotes a *public index* that the encryptor does not mind revealing to all participants in the system. In other words, we define the empty functionality as $f_\epsilon(\mathsf{ind}, m) = (\mathsf{ind}, |m|)$.

Let us specifically consider the case of ABE, where the access policy φ is now considered a public index. In this setting where the access policy does not require protection, we have fairly broad and efficient constructions of secure ABE schemes. Namely, secure ABE schemes exist that support any access policy φ that can be expressed as a boolean formula over the attributes (as in the examples above) [GPSW06, BSW07, OSW07, LSW10, LOS$^+$10, OT10, Wat11]. Going beyond policies expressible as boolean formulas remains a vexing open problem, with the ultimate goal of supporting policies expressible as arbitrary boolean circuits or Turing Machines.

3.2 Non-Public Index

A much more challenging setting arises where we insist that the empty functionality reveals as little as possible, namely $f_\epsilon(x) = |x|$. Here, our current understanding of functional encryption is extremely limited. The state-of-the-art in this setting is limited to the inner-product functionality over prime fields [KSW08, LOS$^+$10, OT10, AFV11]. Because this functionality is somewhat technical, before we describe this functionality more formally, let us briefly discuss some applications: Consider the question of searching on encrypted data, where the data is encrypted based on a public key and stored on a public server [BCOP04]. The security challenge in this setting is to hide the specific nature of the search query from the public sever, while still allowing the public server to send back only data entries that match the search query. The inner-product functionality we describe below allows one to perform such searches based on *disjunction* queries, and more generally searches defined by CNF and DNF formulae, or by checking whether a univariate search polynomial evaluates to zero on a particular input value.

The functionality we consider will be defined over a prime field $\mathbb{F}_p$, where p is a large prime chosen randomly during the setup of the functional encryption scheme. Both messages and keys will correspond to vectors of a fixed arbitrary dimension n over $\mathbb{F}_p$. Let us denote the message by the vector $\mathbf{v}$ and the vector underlying a secret key by $\mathbf{u}$. Then we have:

$$f_{\mathbf{u}}(\ \mathbf{v}\ ) := \begin{cases} 1 & \text{if } \sum_{i=1,\ldots,n} \mathbf{u}_i \cdot \mathbf{v}_i = 0, \\ \perp & \text{otherwise} \end{cases} \tag{2}$$

To see how the above functionality can be applied, consider again the example of disjunction queries: Suppose that a ciphertext is meant to encrypt a single keyword, which we hash down to a value a in our finite field $\mathbb{F}_p$. Then to encrypt this value a, we actually encrypt the vector $\mathbf{v} = (1, a, a^2, \ldots, a^{n-1})$. Now suppose that we need to create a key corresponding to a disjunction query "a_1 OR a_2 OR a_3". We will do so by first considering the polynomial $p(x) = (x - a_1)(x - a_2)(x - a_3)$ and writing it out in standard form as $p(x) = c_0 + c_1 x + c_2 x^2 + c_3 x^3$, where the c_i are the appropriate coefficients. Then we will issue a key for the vector $\mathbf{u} = (c_0, c_1, c_2, c_3, 0, \ldots, 0)$. Glancing at the functionality, we see that indeed our key will match the ciphertext for value a if and only if $p(a) = 0$. In other words, if the value a is a root of our polynomial $p(x)$, which was designed to only have roots at the three values a_1, a_2, a_3 in our desired disjunction. Other special cases of inner-products including conjunctions and range testing functionalities are considered in [BW07].

Unfortunately the exact cryptographic mechanisms by which the [KSW08, LOS$^+$10, OT10] results work are too technically involved to describe here. We encourage the interested reader to look at these works for further technical details.

3.3 Current Limitations

As detailed above, current functional encryption schemes, especially in the non-public index setting, are limited. From a technical standpoint, current techniques for building functional encryption schemes are all based on elliptic curve groups that are equipped with efficiently computable bilinear pairings (that map into the multiplicative structure of a finite field). At a very high level a pairing operation allows for a single multiplication between the exponents of two "source" group elements. The result of a pairing operation, however, is a "target" group for which the operation cannot be repeated.

The reason why we can handle inner products of two vectors is because this operation only requires one parallel call to the multiplication operation, which is what bilinear maps provide. A tantalizing question is whether techniques from lattices, which have been so useful in the context of fully homomorphic encryption [Gen09], can be applied to achieve greater functionality for functional encryption.

3.4 Efficiency

The efficiency of functional encryption systems will vary significantly with the functionality offered and the specific realization. However, we can offer a rough sense of the efficiency of the ABE where the ciphertext is associated with a any access policy φ that can be expressed as a boolean formula over attributes. In current systems, the size of the ciphertext will scale with the size of the boolean formula φ. For example, in [Wat11] a ciphertext consists of two group elements for every leaf node of ϕ and encryption takes three exponentiations for every leaf node. Decryption will require two of the aforementioned pairing operations for each used attribute in the formula. While it is hard to predict how future functional encryption systems may evolve, one might expect at a high level that the number of public key operations required will scale with the complexity of the functionality.

4. FUNCTIONAL ENCRYPTION VS. FULLY HOMOMORPHIC ENCRYPTION

Fully homomorphic encryption (FHE) is arguably the most impressive development in cryptography over the past few years. FHE enables one to compute on ciphertexts in the following sense: given a public key pk, encryptions of messages $x_1, \ldots, x_\ell$ under pk, and the description of a function f as input, anyone can construct an encryption of the message $f(x_1, \ldots, x_k)$. We refer the reader to Gentry [Gen10] for a detailed discussion. A more restricted version of FHE, called *univariate FHE*, allows anyone to construct an encryption of $f(x)$ from an encryption of m for for all *univariate* functions f.

While both FHE and functional encryption support some form of computation on ciphertexts, it is not known how to construct functional encryption from FHE. In fact, FHE does not even seem to imply basic functionalities such as Identity Based Encryption. The reason is that the output of an FHE computation on encrypted data is an encrypted result. In contrast, the output of a functional encryption computation is available in the clear.

To further illustrate this difference between the two concepts consider once again the spam filtering example from the introduction. In that example the spam filter is given a secret key $\text{sk} = \text{sk}[f]$ where f is a function that outputs 1 if an email is spam and 0 otherwise. The key sk lets the spam filter run the spam predicate f on encrypted emails and block encrypted spam. With FHE the spam filter can similarly run the spam predicate f on encrypted email, but the filter only learns the *encrypted* output of the predicate – it does not and cannot learn whether an encrypted email is spam or not. In particular, with FHE the filter can only *tag* an encrypted email with an encrypted tag indicating "spam" or "not spam," but cannot block spam email for the end user. This example illustrates the potential power of functional encryption over FHE. Of course, constructing a fully general functional encryption scheme is still an open problem, whereas FHE is already known to exist.

5. GENERALIZATIONS

We conclude with a few generalizations, variants, and extensions of functional encryption which are motivated in practice.

Delegating Keys. There are several instances where a user might want to delegate a limited set of their capabilities to another user or device. For instance a medical researcher with a secret key that can decrypt raw medical records, might want to distribute to a grad student a key that can only output certain statistics such as averages over the data. As another example, suppose a user is planning to travel with their mobile device, but is concerned that the device might become lost or stolen. Then they might want to copy a key to the device that only decrypts data that was encrypted during the travel time or restrict the key to capabilities related to the business of the trip.

A simple approach is for a user with a key $\text{sk}[f]$ to query the authority for a more restrictive key $\text{sk}[f']$ anytime he wishes to delegate a key for a more restrictive function f'. However, involving the authority in every delegation is cumbersome, exposes an online authority to more risk and won't work if the authority is unreachable. Therefore, we would like the delegation operation to be *autonomous*. Roughly, a user with $\text{sk}[f]$ can create $\text{sk}[f']$ if f' is more limited than the function f — whatever we can learn from f' we can learn from f.

The concept of delegation arose in Identity-Based Encryption [HL02, GS02] and can be realized in Attribute-Based encryption [GPSW06].

Functionality Over Multiple Authorities.

In a standard functional encryption system there is one authority who is responsible for issuing private keys. However, some systems might require more flexibility. Returning to the example of (Ciphertext-Policy) Attribute-Based Encryption, in a standard system one authority will be responsible both for determining which attributes/credentials to issue to each user and for creating the keys.

While this is likely workable for a small to medium scale organization, in many applications we might wish to create policies that span over many trust domains. For instance suppose I wish to encrypt a document for all military personnel who are also members of the ACM. Who should manage this? Using a central authority creates several problems. For one, there will often not be one party who can speak authoritatively for multiple trust domains or organizations. Indeed, one might wish to create a policy that spans organizations that are not even aware of each other. Another core limitation is that a central authority creates a central performance bottleneck and a consolidates trust in one entity. Will two different organizations be able to agree who should be trusted in this role?

Recent work in Decentralized Attribute-Based Encryption [Cha07, LW11] has sought to overcome these limitations. In these systems a user can encrypt according to an ABE policy issued as a formula over attributes issued from different authorities. An interesting direction is to see what other functionalities beyond ABE can arise from the use of multiple authorities in functional encryption systems.

Functional encryption with Public-Key Infrastructure.

Finally, we consider how ideas from functional encryption can be applied to other scenarios. Specifically, consider a scenario where:

- There exists a per-user public key infrastructure, where every user u obtains a secret key $\mathrm{sk}_u[f]$ for some function f_u appropriately chosen by the user, *and* also establishes a public key pk_u unique to the user. This public key should also not leak any information about the function f_u. Such a private and public key is established through an interaction between an authority (CA) and the user.
- Encryptions are always targeted at a specific user's public key pk_u. However, the encryptor does not know the function f_u corresponding to the user, which is hidden by the public key pk_u. At the same time, if a user u obtains an encryption of x under the user's public key pk_u, then decryption allows the user to learn $f_u(x)$, and nothing more. Users should also not be able to obtain additional capabilities by combining secret keys corresponding to different public keys.
- A misbehaving central authority should not be able to decrypt encryptions intended for honest users in the system.

We stress that this scenario is quite different from the functional encryption scenario considered here. One of the key properties of functional encryption is that it does not require public-key directories, thus enabling a variety of applications such as secure storage in the cloud and secure searching on encrypted data. At the same time, this comes at the cost of needing to trust a key generation authority (or a set of such authorities) that is capable of breaking the security of ciphertexts.

This scenario outlined here was considered in recent work [SS10]. There, it is shown that in this setting, called "Worry-Free Encryption," one can support functions (in the non-public index setting) specified by any arbitrary polynomial-size circuit, which is significantly beyond what is possible with general functional encryption. It must be stressed, however, that this setting does not cover motivating applications of functional encryption such as secure storage in the cloud and searching on encrypted data. We refer the reader to [SS10] for more details on this setting.

6. THE FUTURE OF FUNCTIONAL ENCRYPTION

What will functional encryption look like in 10 years? While existing functional encryption systems are already remarkably expressive, the central challenge is to construct a functional encryption system that supports the creation of keys for *any* function f in both public and non-public index settings. If we could create such systems we could imagine embedding anything from arbitrarily complex spam filters to image recognition algorithms into encryption systems. Imagine an encryption system that only lets you view an image if a facial recognition algorithm matches a picture of you to a face in the encrypted image. Moreover, the output of the decryption could show the area immediately surrounding the identified user and blur out the rest of the image.

Current progress on building functional encryption systems has been driven and dominated by the aforementioned tool of groups with bilinear maps. However, as mentioned earlier there are some reasons to suspect that there might be fundamental barriers in realizing more advanced functional encryption systems from this tool. It seems likely that to move forward we will need to search further out. One reason for optimism has been the recent dramatic leap in what we could achieve in homomorphic encryption systems. Hopefully, such a leap will be seen in the not too distant future (perhaps using related techniques) in the realm of functional encryption.

Finally, more applied research is needed to build functional encryption into real-world systems as well as to specify formats for attribute spaces and languages for expressing access policies. Thanks to the expressive power of these systems we hope to see real-world deployments of functional encryption over the next decade. The end result is far greater flexibility in specifying who can and cannot access protected data.

7. ACKNOWLEDGEMENTS

Dan Boneh is supported by NSF, DARPA PROCEED, the Air Force Office of Scientific Research (AFO SR) under the MURI award for "Collaborative policies and assured information sharing" (Project PRESIDIO), a Google Faculty Research Award, and by the Packard Foundation. Amit Sahai is supported by a DARPA/ONR PROCEED award, NSF grants 1136174, 1118096, 1065276, 0916574 and 0830803, a Xerox Foundation Award, a Google Faculty Research Award, an equipment grant from Intel, and an Okawa Foundation Research Grant. This material is based upon work supported by the Defense Advanced Research Projects Agency through the U.S. Office of Naval Research under Contract N00014-11-1-0389. Brent Waters is supported by NSF CNS-0915361 and CNS-0952692, AFOSR Grant No: FA9550-08-1-0352, DARPA PROCEED, DARPA N11AP20006, Google Faculty Research award, the Alfred P. Sloan Fellowship, Microsoft Faculty Fellowship, and the Packard Foundation.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the Department of Defense or the U.S. Government.

8. REFERENCES

Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (h)ibe in the standard model. In *EUROCRYPT*, pages 553–572, 2010.

Shweta Agrawal, David Mandell Freeman, and Vinod Vaikuntanathan. Functional encryption for inner product predicates from learning with errors. In *ASIACRYPT*, pages 21–40, 2011.

Dan Boneh and Xavier Boyen. Efficient selective-id secure identity-based encryption without random oracles. In *EUROCRYPT*, pages 223–238, 2004.

Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In *EUROCRYPT*, pages 506–522, 2004.

Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. In *Proc. of Crypto'01*, volume 2139 of *LNCS*, 2001. full version in SIAM J. of Computing 2003.

John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *IEEE Symposium on Security and Privacy*, pages 321–334, 2007.

Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In *TCC*, pages 253–273, 2011.

Dan Boneh and Brent Waters. Conjunctive, subset, and range queries on encrypted data. In *TCC*, pages 535–554, 2007.

Melissa Chase. Multi-authority attribute based encryption. In *TCC*, pages 515–534, 2007.

Clifford Cocks. An identity based encryption scheme based on quadratic residues. In *IMA Int. Conf.*, pages 360–363, 2001.

Whitfield Diffie and Martin E. Hellman. Multiuser cryptographic techniques. In *AFIPS National Computer Conference*, pages 109–112, 1976.

Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

Craig Gentry. Practical identity-based encryption without random oracles. In *EUROCRYPT*, pages 445–464, 2006.

Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.

Craig Gentry. Computing arbitrary functions of encrypted data. *Commun. ACM*, 53(3):97–105, 2010.

Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, pages 218–229, 1987.

Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *ACM Conference on Computer and Communications Security*, pages 89–98, 2006.

Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*, pages 197–206, 2008.

Craig Gentry and Alice Silverberg. Hierarchical id-based cryptography. In *ASIACRYPT*, pages 548–566, 2002.

Jeremy Horwitz and Ben Lynn. Toward hierarchical identity-based encryption. In *EUROCRYPT*, pages 466–481, 2002.

Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *EUROCRYPT*, pages 146–162, 2008.

Allison B. Lewko, Tatsuaki Okamoto, Amit Sahai, Katsuyuki Takashima, and Brent Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In *EUROCRYPT*, pages 62–91, 2010.

Allison B. Lewko, Amit Sahai, and Brent Waters. Revocation systems with very small private keys. In *IEEE Symposium on Security and Privacy*, pages 273–285, 2010.

Allison B. Lewko and Brent Waters. Decentralizing attribute-based encryption. In *EUROCRYPT*, pages 568–588, 2011.

Rafail Ostrovsky, Amit Sahai, and Brent Waters. Attribute-based encryption with non-monotonic access structures. In *ACM Conference on Computer and Communications Security*, pages 195–203, 2007.

Tatsuaki Okamoto and Katsuyuki Takashima. Fully secure functional encryption with general relations from the decisional linear assumption. In *CRYPTO*, pages 191–208, 2010.

Adi Shamir. Identity-based cryptosystems and signature schemes. In *CRYPTO*, pages 47–53, 1984.

Amit Sahai and Hakan Seyalioglu. Worry-free encryption: functional encryption with public keys. In *ACM Conference on Computer and Communications Security*, pages 463–472, 2010.

Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *EUROCRYPT*, pages 457–473, 2005.

Brent Waters. Efficient identity-based encryption without random oracles. In *EUROCRYPT*, pages 114–127, 2005.

Brent Waters. Dual system encryption: Realizing fully secure ibe and hibe under simple assumptions. In *CRYPTO*, pages 619–636, 2009.

Brent Waters. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. PKC, 2011.

Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *FOCS*, pages 160–164, 1982.

Dan Boneh (dabo@cs.stanford.edu) is a Professor of Computer Science and Electrical Engineering at Stanford University.

Amit Sahai (sahai@cs.ucla.edu) is a Professor of Computer Science at the University of California, Los Angeles (UCLA).

Brent Waters (bwaters@cs.utexas.edu) is an Assistant Professor of Computer Science at the University of Texas at Austin.